Idea: Supporting Policy-Based Access Control on Database Systems

Jasper Bogaerts, Bert Lagaisse, and Wouter Joosen

iMinds-DistriNet, KU Leuven, 3001 Leuven, Belgium
first.last@cs.kuleuven.be

Abstract. Applications are increasingly operating on large data sets. This trend creates problems for access control, which in principle restricts the actions that subjects can perform on any item in that data set. Performance issues therefore emerge, typically for operations on entire data sets. Emerging access control models such as attribute-based access control do meet their limitations in this context. Worse, few solutions exist that addresses performance problems while supporting separation of concerns. In this paper, we present a first approach towards addressing this challenge. We propose a middleware architecture that performs policy transformations and query rewriting for externalized policies to optimize the access control process on the data set. We argue that this offers a promising approach for reducing the policy evaluation overhead for access control on large data sets.

Keywords: Access control, policy-based access control, databases, attribute-based access control

1 Introduction

Applications are increasingly operating on large data sets. This is especially true for multi-tenant software-as-a-service (SaaS) applications, in which tenant organizations access a shared, typically web-based application hosted by a provider [12].

Such data must be protected. One important security measure to protect data is access control, which restricts actions performed by a subject (e.g., user) on an object (e.g., resource). A typical approach to realize this is to externalize an access control policy from the application and evaluate it each time a subject performs a request to the application [20, 22], a technique commonly referred to as Policy-Based Access Control (PBAC). This supports separation of concerns [7] and enables tenants in multi-tenant SaaS applications to specify their own policy without service interruption [8].

One challenge for policy-based access control is to enforce it for operations on a large data set. An *operation* comprises of the same action that is performed on each element of the data set. For example, when subjects perform a search on a database, only the elements (or *objects*) to which they are entitled should be returned. This involves a policy evaluation for the *view* action on the objects.

A naive approach to such issue is to serially evaluate the access control policy for each data item returned by the search query and filter the results. However, this can involve considerable evaluation overhead [21], especially for policies specified according to emerging access control models such as attribute-based access control [11]. On the other hand, while most database management systems support some form of efficient access control enforcement, they do not provide a solution to this problem that both scales in terms of the number of users and organizations, and meanwhile provides separation of concerns.

In this paper, we approach this issue by merging policy transformation and query rewriting techniques to optimize operations on large data sets. These techniques should be employable in a manner that is transparent to the application developer, thereby accomplishing a separation of concerns. This paper provides a first analysis of the approach that combines these techniques for this purpose, and discusses challenges that need to be addressed to realize such an approach.

Organization. Section 2 provides a scenario and motivates the requirements. This section also elaborates on related work. Section 3 describes the middleware that we propose to mitigate the problem. Section 4 discusses the proposed solution further, and elaborates on the challenges. Section 5 concludes the paper.

2 Motivation

This paper describes how policy transformation and query rewriting techniques can be combined to enforce access control policies for operations on large data sets. Although several such operations can be performed, such as batch insertions or updates, our main focus will be to enforce this for search operations.

Fig. 1. The document management application enables tenants to specify policies.

To illustrate this issue, consider a document management SaaS application, that manages millions of business documents for multiple tenants, as shown in Figure 1. These tenants can each specify their own policies to restrict their respective affiliates. For example, they could permit read access to the creator of a document, and to all members of his/her department (e.g., accounting). The affiliates should also be restricted access based on policies that were specified by the application developers. In general, for example, such an affiliate cannot access documents of another tenant. A tenant can not be restricted access to its own documents through policies of another tenant [8].

When affiliates of a tenant search for documents in the document management application, they must only be returned the documents they are entitled to. This involves enforcement of application policies (e.g., subjects can only access documents of their tenant) and tenant policies (e.g., subjects of the accounting department can view paychecks).

2.1 Requirements

Besides functional requirements previously indicated in the scenario, the solution must support several additional characteristics:

Transparency. The middleware must integrate seamlessly with the existing application, and respect the principle of separation of concerns. More particularly, the application developer should not take into account access control when writing queries.

Support for application-level policies. The middleware must take into account policies that reason about application concepts. An access control policy is *safe* if it only refers to concepts that exist in the application or, alternatively, refers to the subject that performs actions on the application. Similarly, a query is safe if it corresponds to the underlying database schema.

Support for tenant policies. The middleware must take into account policies specified by tenants. Since tenants provide untrusted input in the policies, we must ensure that they are *secure*, i.e., that they do not escalate privileges over provider policies, nor are they vulnerable to injection attacks.

Support for expressive policies. The middleware should support expressive policies. This enables tenants to specify fine-grained access constraints.

Performance. The middleware should reduce the access control evaluation overhead for operations on large data sets. The overhead that is introduced by the middleware itself must be minimized.

2.2 Related work

A lot of related work has been performed in the domain of database access control. Many traditional database management systems employ views, stored procedures and access control lists to restrict access for individual subjects [3]. However, many such techniques assume a two-tier architecture, which has no use when the application performs a query on behalf of a subject, as is the case in multi-tier architectures that are common today [19]. As a result, such access control techniques cannot be effectively enforced at the database at a granularity level that exceeds the application. This is also the case for many techniques that employ query rewriting to optimize access control [5, 10, 15, 18].

One technique that mitigates this issue is Virtual Private Database (VPD, [1]). VPD supports application identification, as a complement to subject identification, to enforce access control policies that are specified in the DBMS. While this supports the specification of views and queries that are aware of individual subjects, VPD requires policies to be specified at the database management system.

This does not adhere to the principle of separation of concerns. Moreover, it requires the policies to restrict in terms of database operations (e.g., insertion and selection), whereas application actions may involve multiple such operations.

Opyrchal et al. [16] have addressed a similar issue to our goal by enforcing CPOL policies for databases. Their system first checks whether a query is permitted and evaluates the policy for each returned element if it is. However, their method involves only limited query rewriting and no policy transformations. Consequently, the system does not scale when large data sets are involved, especially if many elements are returned. By performing query rewriting and policy transformation, our approach is able to reduce data sets on which access control is performed, and optimizes access control evaluation for the remaining sets.

Axiomatics data access filter [2] also provides a solution that enforces policy-based access control on databases for attribute-based policies. However, it does not provide sufficient constraints to ensure safe and secure queries in the light of, among others, multi-tenant applications.

Cook et al. [6] focus on the safety of composing queries. They propose a method that effectively restricts developers from specifying queries that do not correspond to the application domain. This is complementary to our work, but it does not focus on the middleware that performs safe query rewriting.

This work relates to that of access control and usage control enforcement techniques. Notably, in [17], Pretschner et al. present an architecture that is capable of enforcing usage control policies in a distributed fashion. A similar approach is taken in [9] to ensure access control enforcement is decentralized. While this relates to our objective to speed up enforcement for access control, our approach does not perform distributed evaluation to achieve this.

3 Approach

In order to meet the requirements that were introduced in Section 2, we propose a middleware that transparently performs access control for operations on large data sets. This section outlines the middleware architecture. The next section discusses the challenges and motivates its feasibility.

The middleware is embedded in a database abstraction layer, and intercepts every query that is performed on the database. The database abstraction layer provides an abstraction over data access and integrates technologies such as JDBC[1] or object-relational mappers (ORM) such as JPA[2] to hide database-specific complexity. The middleware intercepts the query and determines the objects on which the operation can be performed before executing it on them.

In order to support expressive policies, the middleware supports a XACML-like policy language. XACML [14] provides a tree-structured, attribute-based policy language in which policies can themselves contain other policies, thus

[1] Java Database Connectivity, see also `http://www.oracle.com/technetwork/java/overview-141217.html`.

[2] Java Persistence API, see also `http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html`.

forming a policy *tree*. Applicability of both policies and rules are determined during the evaluation, in which a *target* expression indicates applicability for a policy, and a *condition* expression indicates applicability for a rule. If a rule is applicable, its corresponding decision (i.e., permit or deny) is taken into account. Expressions compare attributes assigned to subjects, objects, actions and environment with each other and concrete values in order to determine the applicability. Combining algorithms (e.g., first applicable, permit overrides) provide conflict resolution when multiple policies or rules are applicable.

3.1 Design

The middleware combines policy transformation and query rewriting techniques to optimize access control evaluation for large data sets. Figure 2 outlines the architecture. The access control middleware introduces two components: the policy transformer and the query translator.

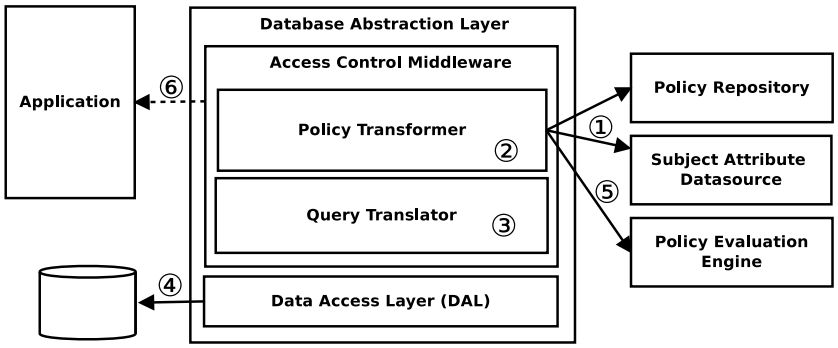

Fig. 2. The middleware employs policy transformations and query rewriting to reduce evaluation overhead for access control.

The policy transformer is responsible for retrieving, validating and transforming the policy. This results in a transformation of the original policy to a reduced form with the same semantics. The component achieves three goals. First, it substitutes all attributes that are not assigned to the object, and as a consequence, remain fixed for every element of the query that must be evaluated against the policy. Second, it prunes the policy by omitting the rules that are never applicable, thereby reducing evaluation time. Third, it reduces the policy and its tree structure in such a way that it simplifies the translation to a query.

The query translator takes as input the transformed policy, and translates it to a query that can be executed on the database. In order to support a wide range of underlying database systems, we do not require the policy to be incorporated in the resulting query entirely. Rather, the translator may select and translate a *partial* policy that can significantly reduce the data set on which the transformed

policy must be evaluated. This enables the middleware to be used for database systems that have a constrained query language, which is common especially among NoSQL systems. Moreover, it supports translation to only queries that can be performed without significant overhead.

Figure 2 illustrates how the middleware handles the access control process. The access control process for search operations consists of six steps:

1. **Policy retrieval and validation.** The policy transformer retrieves and combines all policies that are relevant for the acting subject. Next, it validates that the concepts that are referred to in the policy apply to the application. This is especially a problem when attribute-based policies are involved, because attribute-based policies require an understanding of the properties associated with the subjects, objects, and actions of the application domain. Validation ensures that no errors can occur in the evaluation in this regard. Moreover, it can ensure that queries that are rewritten are safe (i.e., they refer only to application concepts) and secure (i.e., they do not lead to privilege escalation) because they are translated from a sanitized set of expressions.
2. **Policy rewriting.** The policy transformer retrieves all relevant attributes of the acting subject, action and environment. This information is leveraged to prune the policy to contain only relevant rules. This stems from the observation that for each object, the subject and action will remain the same and may lead to rules that are always (in)applicable for the search operation. Such rules enable pruning of the policy. To do this, the middleware substitutes the attributes with concrete values and determines which rules and policies can be pruned. This can reduce policy evaluation time and reduce the query that is generated based on the access control policy later in the process. For instance, consider that an accountant searches all documents in the document management platform that match a certain search term. If the policy also includes rules that target other roles or actions, they can be pruned for the transformation.
3. **Policy to query translation.** In this step, the transformed policy is translated to a query. Note that the query should evaluate only concepts of the objects on which the search operation is performed, due to the substitution of attributes in the previous step. The query translator could select only a *partial* policy to translate to due to functional or performance constraints of the underlying database. Such a query must significantly reduce the size of the data set to optimize the access control evaluation process. For instance, consider again the example scenario of the previous step. Consider also that the policy states that accountants can only read financial documents that were created in the last year. The query translation could select only a part of this rule, e.g., that only financial documents may be read, to cope with constraints of the query language of the underlying database. This could already significantly reduce the resulting data set on which serial policy evaluation must be performed, because other types of documents for which the policy would evaluate to a deny decision are already filtered out.

4. **Query.** The query translator composes a query that takes into account both the original request parameters (e.g., search parameters) and the access control policy (translated in the previous step). Next, it retrieves all objects that satisfy the composed query from the underlying database.
5. **Policy evaluation.** The policy transformer evaluates the previously transformed policy for each object that resulted from the query. This determines on which objects the subject is entitled to perform the search operation. For instance, consider the translation of a partial policy in the example of the third step. The previously transformed policy can be evaluated against the result set to enforce that the documents were created in the last year. Note that this step is redundant when the policy is fully translated to the query, and can be skipped in such a case.
6. **Result.** Finally, the resulting data set is returned to the subject.

4 Discussion

With the architecture presented in Section 3, we intend to significantly reduce the policy evaluation overhead for operations on large data sets. However, in order to do this, several challenges need to be addressed.

The middleware optimizes access control for large data sets through policy transformation and query rewriting. Both can introduce an overhead. On the one hand, policy transformation may introduce a processing overhead, and still requires considerable overhead when the transformed policy is evaluated against a large data set. On the other hand, performing a query that was rewritten according to a policy may also introduce an overhead when applied to a large data set. Consequently, a balance must be found in determining to what extent the query is rewritten. This is complicated by the variability of the underlying database schema and the query languages that are supported by the database system. This is a considerable challenge for future work, especially when NoSQL systems must be supported. In general, these systems are identified by constraints in their query language and a potentially large cost for performing certain types of queries. Whenever a policy is only translated partially into a query, this requires an evaluation of the transformed policy over the objects in the data set that results from this query. This is performed in the fifth step of the process. Because some of the expressions of the policy are already included as part of the query, the policy could be further transformed to omit these expressions and hence avoid redundant evaluation. When the policy can be translated fully into a query, this post-evaluation step can be omitted altogether.

While we focused on constraining search operations, a similar approach could be performed for write operations (e.g., batch updates). This would involve a step that filters out objects for which the operation is not permitted prior to performing the query, if the policy can not be translated fully to a query.

The strategy introduced in this paper requires that both database and middleware preside in the same security domain. Else, the solution would be subject to data leaking for objects that were not filtered by the query, but are withheld by the policy evaluation.

In order to determine the feasibility of the approach presented in this paper, we have induced a prototype that is capable of handling policy transformations for the STAPL [13] policy language (which closely resembles XACML [14]) and performs query rewriting for SQL-compliant database systems. To ensure the safety and security of the queries that are generated in our approach, validation techniques can be employed. Safety validation can be performed through matching the referred attributes to the application domain concepts [4]. This is done through a separate artifact that describes the properties of the subjects, objects and actions associated with the application domain and how they map on the database schema. This artifact can be extracted automatically from the application code. Security validation intends to prevent privilege escalation through queries generated from custom policies, and can also employ this artifact in combination with whitelisting techniques for the expressions of the policy to determine whether they can be translated to a query securely. The initial prototype indicates the feasibility of the approach, and a thorough evaluation of the performance will be presented in future work.

5 Conclusion

This paper has presented an initial step towards a middleware than can transparently enforce access control for search operations on large data sets. Evidently, many challenges remain. These include the way that the middleware handles variability of the underlying database schema and analyzing the performance issues that large policies may introduce on the translated query. However, we believe that such problems can be mitigated.

We are convinced that the middleware presented in this paper can significantly reduce the overhead introduced by performing access control for large data sets, and should be further researched in future work. This would enable policy-based access control to be enforced on both the application and the database, effectively supporting a separation of concerns.

Acknowledgments

This research is partially funded by the Research Fund KU Leuven, and by the EU FP7 project NESSoS. With the financial support from the Prevention of and Fight against Crime Programme of the European Union (B-CCENTRE).

References

1. Oracle Virtual Private Database (VPD). `http://docs.oracle.com/cd/B28359_01/network.111/b28531/vpd.htm`. Accessed: 2015-09-02.
2. Axiomatics. Data Access Filter (ADAF). `http://www.axiomatics.com/solutions/products/authorization-for-databases/197-axiomatics-data-access-filter-adaf.html`, 2015. [Online; accessed 2-Oct-2015].

3. E. Bertino and R. Sandhu. Database security-concepts, approaches, and challenges. *Dependable and Secure Computing, IEEE Transactions on*, 2(1):2–19, 2005.
4. J. Bogaerts, M. Decat, B. Lagaisse, and W. Joosen. Entity-Based Access Control: Supporting More Expressive Access Control Policies. In *Proceedings of the 31st Annual Computer Security Applications Conference*, 2015.
5. B. Carminati, E. Ferrari, J. Cao, and K. L. Tan. A framework to enforce access control over data streams. *ACM TISSEC*, 2010.
6. W. R. Cook and S. Rai. Safe query objects: statically typed objects as remotely executable queries. In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, pages 97–106. IEEE, 2005.
7. B. De Win, F. Piessens, W. Joosen, and T. Verhanneman. On the importance of the separation-of-concerns principle in secure software engineering. In *Workshop on the Application of Engineering Principles to System Security Design*, 2002.
8. M. Decat, J. Bogaerts, B. Lagaisse, and W. Joosen. Amusa: middleware for efficient access control management of multi-tenant SaaS applications. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015.
9. R. Gay, J. Hu, and H. Mantel. CliSeAu: Securing Distributed Java Programs by Cooperative Dynamic Enforcement. In *Information Systems Security*. Springer, 2014.
10. E. Grummt and M. Müller. Fine-Grained Access Control for EPC Information Services. In *The Internet of Things*. Springer, 2008.
11. V. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone. Guide to Attribute Based Access Control (ABAC) Definition and Considerations. *NIST Special Publication*, 2014.
12. P. Mell and T. Grance. The NIST definition of cloud computing. *NIST*, 2009.
13. J. Moeys and M. Decat. Simple Tree-structured Attribute-based Policy Language (STAPL). `https://github.com/stapl-dsl`, 2015. [Online; accessed 2-Oct-2015].
14. OASIS. eXtensible Access Control Markup Language (XACML) Standard v3.0. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf`, 2013.
15. L. E. Olson, C. A. Gunter, W. R. Cook, and M. Winslett. Implementing reflective access control in SQL. In *Data and Applications Security*. Springer, 2009.
16. L. Opyrchal, J. Cooper, R. Poyar, B. Lenahan, and D. Zeinner. Bouncer: Policy-Based Fine Grained Access Control in Large Databases. *International Journal of Security and Its Applications*, 2011.
17. A. Pretschner, M. Hilty, and D. Basin. Distributed usage control. *Communications of the ACM*, 2006.
18. S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *SIGMOD Conference on Management of data*. ACM, 2004.
19. A. Roichman and E. Gudes. Fine-grained access control to web databases. In *Symposium on Access control models and technologies*. ACM, 2007.
20. P. Samarati and S. Vimercati. Access Control: Policies, Models, and Mechanisms. In *Foundations of Security Analysis and Design*, pages 137–196. 2001.
21. F. Turkmen and B. Crispo. Performance evaluation of XACML PDP implementations. In *Workshop on Secure web services*. ACM, 2008.
22. J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. RFC 2904: AAA Authorization Framework, August 2000.